VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2018 Jakub Čajka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jakub Čajka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: IDC CEMA, s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

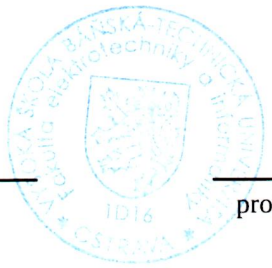
Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Konzultant bakalářské práce: Ing. Marek Balgar

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 11. dubna 2018 ..... Čajka .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

IDC CEMA s.r.o.
Malé náměstí 13, 110 00 Praha 1
IČ: 26482347, DIČ: CZ26482347

V Ostravě 11. dubna 2018

.....

Rád bych na tomto místě poděkoval především společnosti International Data Corporation, která mi umožnila pracovat a vzdělávat se v prostředí se zkušenými vývojáři. Dále pak bych také rád poděkoval Ing. Markovi Balgarovi, který mi byl během mé práce neustálou oporou a vynikajícím týmovým vůdcem. V neposlední řadě patří také můj dík panu Ing. Davidovi Ježkovi, Ph.D., jehož rady mi v průběhu psaní mé bakalářské práce velice pomohly.

Abstrakt

Cílem této bakalářské práce, je popsat mé působení ve společnosti International Data Corporation (IDC). Při nástupu do této společnosti jsem byl přidělen do týmu Internal Tools Group (ITG), který vyvíjí interní programy. A to za účelem usnadnění práce našim analytikům, administrace uživatelů, nebo shromažďování dat. Na začátku mé pracovní činnosti jsem pracoval na automatických testech jedenoho z našich projektů (Edison). Dále jsem pracoval na pozici aplikační developer pro ostatní projekty. V této práci popíšu úlohy, které jsem při své činnosti prováděl, problémy, které se při tom vyskytly a jak jsem tyto problémy řešil.

Klíčová slova: International Data Corporation, IDC, Internal Tools Group, ITG, Data-Warehouse, Edison, odborná praxe

Abstract

The aim of this bachelor thesis is to describe my work at the International Data Corporation (IDC). At the begging of my work in the company, I was assigned to the Internal Tools Group Team (ITG), which develops internal programs. For the purpose of easier work process for our analytics, administration of users or gathering of data. At the beginning of my career I worked on automation tests for one of our project (Edison). Then I worked as an application developer for other projects. In this thesis I am going to describe tasks which I did, problems I encountered in the process and how I solved this issues.

Key Words: International Data Corporation, IDC, Internal Tools Group, ITG, Data-Warehouse, Edison, professional practice

Obsah

Seznam použitých zkratek a symbolů

API	–	Application programming interface
IDE	–	Integrated Development Environment
JDK	–	Java SE Development Kit
VCS	–	Version Control System
IDC	–	International Data Corporation
AWS	–	Amazon Web Services
ITG	–	Internal Tools Group
MVC	–	Model View Controller
QA	–	Quality Assurance
PROD	–	Production
DOM	–	Document Object Model
DTO	–	Data Transfer Object

Seznam obrázků

1 Úvod

Tématem bakalářské práce je popis mé činnosti aplikačního programátora ve firmě IDC. Důvod, proč jsem si vybral vypracovat svou bakalářskou práci touto formou je, že, dle mého názoru, mi poskytne cenné zkušenosti, jako například práce na reálných projektech, které budu moct dále uplatnit.

Na začátku praxe jsem pracoval jako vývojář automatických testů pro aplikaci Edison. Hlavní účel této aplikace je zprostředkování zobrazení dat našim analytikům ve formě tabulek nebo grafů. Tyto obsahují automatické přihlášení do aplikace. A jejich úkolem je automaticky provádět úlohy podle scénáře, které mají otestovat určité prvky v aplikaci.

Poté jsem pracoval jako aplikační developer pro zbylé projekty, který má náš pracovní tým pod sebou. Mým hlavním projektem, na kterém jsem pracoval, byly Interakce a Data-Warehouse. Úlohy byly od menších oprav chyb, až po větší přidání funkcionality. V této bakalářské práci popíšu ty zajímavější z nich.

2 Zaměření firmy a pracovní nasazení studenta

2.1 O firmě

International Data Corporation (IDC) [1] je přední poskytovatel průzkumů trhu, poradenských služeb a to jak pro informační technologie nebo také pro telekomunikační technologie. IDC zaměstnává více než 1100 analytiků, kteří se podílejí na tvorbě produktů zaměřujících se převážně na výzkum globálních, regionálních a lokálních trhů. V České republice má IDC dvě pobočky v Praze a Ostravě. Obě pobočky jsou zaměřeny na vývoj interního software pro potřeby firmy. V týmu Internal Tools Group máme portfolio dvanácti projektů, které jsou pod naší správou. Patří mezi ně například Data-Warehouse, který shromažďuje data z ostatních projektů a zobrazuje je v přehledné formě. Nebo také Interakce, který je stále ve vývoji, ale v budoucnu v něm budou moct analytici, kteří prodávají produkty naším zákazníkům, zaznamenávat dohody mezi nimi a zákazníkem.

2.2 Pracovní zařazení studenta a pracovní náplň

Ve firmě jsem pracoval ze začátku jako vývojář automatických testů. Mou hlavní náplní v té době bylo tvořit automatické testy v Javě pomocí frameworku Selenium. Později jsem pracoval jako vývojář aplikací, kde mou hlavní úlohou byl vývoj na ITG projektech. Mezi časté úkony patřil vývoj nové funkcionality na některém z projektů podle nově vznikajících potřeb uživatelů. Dále také integrace dat mezi systémy, nebo opravy chyb, které se během vývoje našli.

ITG tým používá a snaží se při vývoji uplatňovat agilní metodiku SCRUM. Znamená to, že každý den se provádí rychlé porady, kde se probírá to, na čem každý pracuje. Včas se tím zjistí, zda se nikdo ve své práci na něčem nezasekl nebo nepotřebuje pomoct.

Jedna pracovní iterace trvá dva týdny, ve kterém jako tým zvládáme 18 příběhových bodů. Story pointy je jednotka složitosti, kterou přidáváme úlohám při jejich analýze. Na konci pracovní iterace vždy máme retrospektivu, kde probíráme co se stalo v předešlém sprintu, co se nám líbilo, nelíbilo nebo co bychom měli změnit/zlepšit.

Je u nás samozřejmostí, že každá úloha musí projít přes kontrolu kódu. Tam ji zkontroluje další programátor, zda odpovídá již zavedeným pravidlům. Pak se přesune k testerovi na QA k otestování.

2.3 Pracovní prostředí

IDC sídlí ve Vědecko-technologickém parku Ostrava, kde má své kanceláře. V těchto kancelářích pracuje celkově 5 týmů, kde v každém je průměrně 10 lidí. Každý zaměstnanec dostane svůj notebook, na kterém pracuje ať už klasicky v práci, nebo po domluvě z domu.

Komunikace uvnitř firmy probíhá pomocí nově zavedeného klienta Microsoft teams. Tímto způsobem komunikují programátoři mezi sebou. Když je potřeba komunikovat s obchodními

partnery, kteří jsou převážně v Amerických státech, používáme emailového klienta, nebo verze Skype pro společnosti.

3 Harmonogram bakalářské práce

3.1 **1. - 2. týden** Seznámení se s prostředím a technologiemi firmy

- Během prvních dvou týdnů jsem se začleňoval do denního chodu firmy. Byly mi vysvětleny základy obchodní logiky v této firmě a byl mi zprostředkován výcvik používaní technologií, se kterými se budu denně setkávat. Mí noví kolegové v týmu mi během prvních dvou týdnů pomáhali s počátečním nastavením interních systémů. Ty byly nezbytné pro mou nastávající práci.

3.2 **3. týden - 2. měsíc** Vývoj automatických testů

- K lepšímu pochopení obchodní logiky, jsem vyvíjel automatické testy pro náš interní systém Edison. Ten se zaměřuje na zobrazení grafických dat v přehledné formě uživateli. Díky tomu jsem musel lépe pochopit interní obchodní logiku. Pak jsem byl schopen testovat správnost zobrazených dat a následně tuto kontrolu zautomatizovat.

3.3 **3. měsíc - 6. měsíc** Práce na všech interních projektech

- Začátkem třetího měsíce jsem se začal spolupodílet na ostatních projektech. Celkově má náš tým pod sebou přes deset projektů, na kterých se provádí aktivní vývoj nebo údržba. Má práce se odvíjela od oprav menších chyb nebo přidání nové funkcionality do jednotlivých projektů až po návrh architektury databáze, refaktor funkcí nebo integrace dat mezi systémy.

3.4 **7. měsíc - 12. měsíc** Zařazení do podpory na vybraných projektech

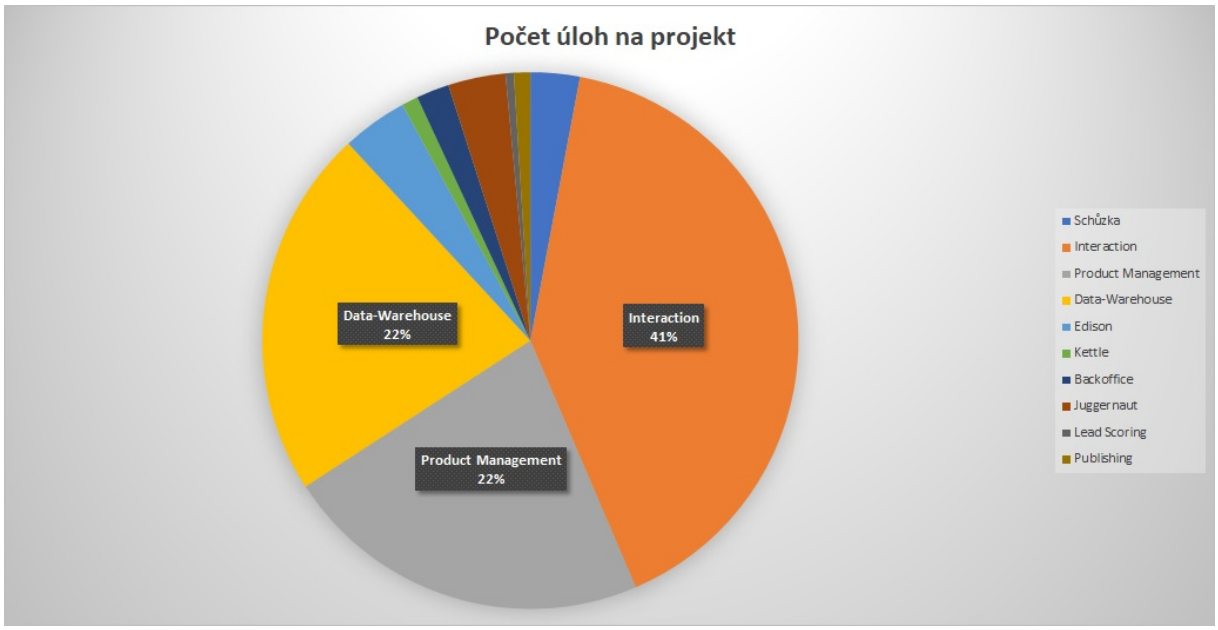
- Po sedmi měsících ve firmě jsem také začal také vyřizovat požadavky uživatelů, pokud mají problém s určitou funkcionalitou, nebo jim něco nefunguje tak, jak by dle specifikací mělo. To zahrnovalo komunikaci v anglickém jazyce přes e-mail nebo mluveným slovem přes interní komunikační kanál. Tyto problémy měli prioritu a ode mě se očekávalo, že dokážu najít příčinu, proč se tento problém vyskytl a posléze zajistit i jeho opravu.

3.5 Přehled úloh vypracovaných na projektech

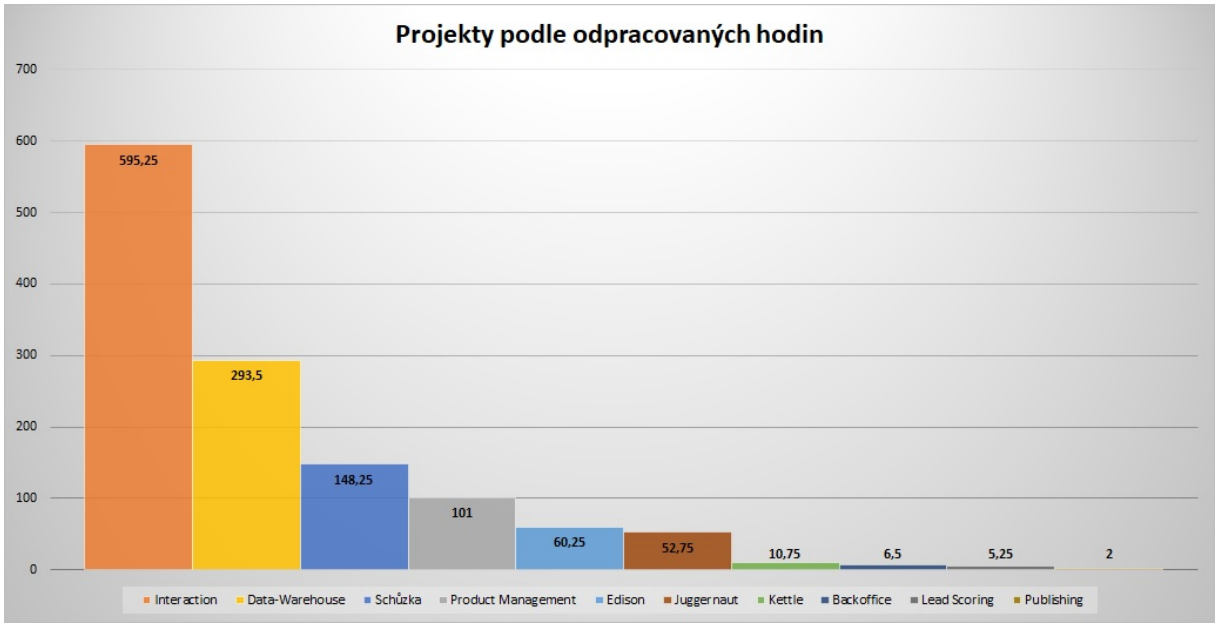
Během své práce jsem pracoval na několika projektech. Na obrázku 1 lze vidět, na kolika úlohách jsem z celkového počtu 202 úloh pracoval. Jsou to jednotlivé projekty za období posledního roku.

3.6 Přehled odpracovaných hodin

Na obrázku 2 lze vidět, kolik hodin z celkového počtu 1275,5 hodin za poslední rok jsem odpracoval na jednotlivých projektech.



Obrázek 1: Poměr úloh k jednotlivým projektům



Obrázek 2: Počet odpracovaných hodin na jednotlivých projektech

4 Použité technologie a nástroje

Mou praxi si nedokážu představit bez níže uvedených technologií a rámců. Uvádím zde alespoň jejich krátký popis.

4.1 Java

Java [2] [3] je objektově orientovaný jazyk. Původně vyvinuta James Goslingem ve firmě Sun Microsystem. Při vývoji aplikace v Javě se aplikace vytváří jen jednou pro Java Virtual Machine (JVM). Tím se docílí, že je aplikace připravena běžet na různých systémech a to počínaje od Windows, přes MacOs až po různé distribuce Linuxu.

Momentálně nejnovejší verze je 9, která přichází s řadou vylepšení jako například rozšíření populárního Stream API z verze 1.8 (přetížení metody iterate) nebo nové tovární metody pro kolekce.

4.2 Javascript

Javascript [4] je scriptovací programovací jazyk, který dovoluje vytváření dynamicky aktualizujícího se obsahu webových stránek, přehrávání videa nebo zvuku nebo animace webových komponent.

Kód provádí tzv. javascript interpreter, který má většina dnešních prohlížečů již v sobě. Javascript díky tomu nevyžaduje žádnou kompilaci a kód se provádí v podstatě tak, jak byl napsán.

Mít část logiky v javascriptu může být výhoda. Tato logika se bude vždy provádět nad procesorem uživatele a nebude tak vytěžovat prostředky serveru a může tak zmenšit vytížení webového serveru.

4.3 Pentaho Spoon

Pentaho Spoon dovoluje uživatelům pomocí jednoduchého grafického rozhraní integraci mezi různými databázovými zdroji. Má mnoho již předprogramovaných funkcí, které slouží k transformaci získaných dat, ale má také JavaScript interpreter, který lze použít na složitější problémy.

Základem je kettle job, který si lze představit jako složku, do níž se poté vkládají kettle transformace. Důležité je, že v tomto jobu se jednotlivé kroky provádějí sériově.

Jako druhým pilířem je kettle transformace, do které už se vkládají integrační kroky. Lze si to představit jako výběr dat nad Oracle databázi a po té vložení dat do PostgreSQL databáze. V transformaci se všechny kroky provádějí paralelně najednou. Proto není vhodné mít v jedné transformaci smazání tabulky a zároveň vložení do této tabulky, neboť může dojít k uzamčení probíhajících připojení. V tomto případě by bylo vhodné smazaní tabulky a vložení rozdělit do dvou transformací a spojit je v jeden kettle job krok, který zpracovává kroky sériově.

4.4 Maven

Maven [5], slouží jako uniformní systém k sestavení především java projektů. Definuje, jak bude výsledný zkompilovaný projekt vypadat.

Pomáhá k udržování závislostí mezi použitými rámci a zjednodušuje přidávání nových knihoven.

Také dovoluje pomocí profilů definovat různé proměnné pro různé prostředí, typicky například QA nebo PROD.

4.5 Flyway

Flyway [6] je plugin, který dovoluje vytváření verzovacích skriptů. Ty pak mění nebo vytvářejí strukturu databáze.

Jeho výhodou je, že při jeho použití máte k dispozici skripty, které v případě potřeby lze použít pro vytvoření kompletní struktury dané databáze.

4.6 Querydsl

Queryds [7] je Java rámec, který dovoluje generování typově bezpečného kódu podobnému SQL v jazyce Java.

Při sestavování projektu pomocí rámce Maven, Querydsl na základě databázové struktury vygeneruje Java objekty. Tyto objekty se poté používají k sestavování dotazů.

Právě generování objektů a specifický přístup k psaní dotazů navazující na tyto objekty lze považovat jako jeho hlavní výhodu. Díky tomu lze totiž při sestavování těchto dotazů používat všechny výhody jazyka Java, popřípadě jiných rámců.

Zároveň je to také ale nevýhoda, protože v případě, že již máte hotový dotaz v jazyce SQL, nutí vás Querydsl přepsat tento dotaz na jeho vygenerované databázové entity.

4.7 Spring

Spring [8] rámec poskytuje komplexní programovací model pro moderní Java internetové aplikace. Spring od základu podporuje MVC model. Tento model je dobře konfigurovatelný díky rozhraním a dokáže se přizpůsobit mnoha zobrazovacím technologiím jako je například JSP, Velocity nebo Tiles. Místo MVC lze také použít jiné rámce.

Spring je jednoduchý, jeho velikost je v jednotkách MB a také jeho režijní náklady jsou téměř zanedbatelné.

Díky jdbc vrstvě, spring nabízí hierarchii vyjímek, která zjednodušuje strategii jejich zachycování.

4.8 GIT

Git [9] je veřejně dostupný distribuovaný verzovací systém. Slouží k lepší spolupráci ve vývoji jednoho nebo více projektů. Mezi nejpoužívanější systémy patří například GitHub nebo GitLab.

Právě kvůli decentralizované podobě gitu, je velice atraktivním nástrojem pro většinu týmů tvořící nějaký společný projekt. Nejenže každý, kdo na nějakém projektu s někým spolupracuje, má vlastní kopii na svém počítači, ale je také uložena na webu. Touto decentralizací lze dosáhnout vysoké úrovně zabezpečení.

Samotný verzovací systém napomáhá při spolupráci jednoduše spojit dvě rozdílné verze jednoho projektu. Zároveň veškeré změny, které kdy byly vytvořeny, jsou přístupné a lze se k nim jednoduše vrátit.

4.9 PostgreSQL

PostgreSQL [10] je open-source objektově relační databázový systém. Je vysoce škálovatelný, jak pro velikost dat, které zvládne zpracovávat, nebo množství uživatelů, kteří mohou najednou k databázi přistoupit.

Funkce PostgreSQL databáze zahrnují například primární klíče, cizí klíče s restrikcemi a kaskádovými aktualizacemi/mazáním, omezení jedinečnosti nebo nenulové omezení.

PostgreSQL také zahrnuje GIST indexování což je vyhledávací systém, který sjednocuje různé vyhledávací a třídící algoritmy jako je například B-tree nebo B+-tree. Poskytuje také rozhraní pro vytváření vlastních datových typů ale také způsob, jak mezi nově vytvořenými typy vyhledávat.

4.10 Sonar

Sonar [11] je systém kontroly nejen syntaxe, ale i funkčnosti. Provádí nepřetržitou inspekci kódu, tím předchází zanesení nových chyb a nebo upozorňuje na opravu už stávajících chyb.

Udržuje soubor pravidel, kterými se programátor má řídit. Buď aby nevytvořil nespolehlivý kód, nebo aby dodržoval konvence, které si jeho tým nastavil pro přehlednost a srozumitelnost kódu.

Sonar má také plugin do různých vývojových prostředí jménem SonarLint. SonarLint provádí nepřetržitou kontrolu pravidel, kterou neustále aktualizuje ze Sonar serveru. Díky tomu může programátor vidět analýzu svého kódu v reálném čase, bez nutnosti odesílat své změny na server, kde by tahle analýza proběhla.

4.11 Jira

Jira [12] je proprietární systém od firmy Atlassian, který slouží ke sledování úloh nebo správě projektů. Nabízí nástroje pro tvorbu reportů nebo vypisování statistik, které jsou užitečné například při plánovaní budoucích projektů.

Systém zároveň slouží ke komunikaci s vývojáři a to formou vytváření úloh, které jsou poté zařazeny do sprintu, na němž vývojáři pracují. Systém se dá přizpůsobit dle potřeby a to úpravou tabule týmu nebo personalizace stavů pro proud úloh.

4.12 Jenkins

Jenkins [13] je open-source server, který pomáhá automatizovat úlohy, díky čemuž dokáže neustále se opakující procesy zrychlit a zjednodušit. Jenkins spravuje a kontroluje proces vývoje a to včetně spouštění kompilace kódu, dokumentace, testů, nasazení na různá prostředí nebo vytváření analýz.

Jenkins je psaný v Javě a je tudíž jednoduché jej spustit na většině operačních systémů.

4.13 Selenium

Selenium [14] je soubor nástrojů, které jsou určeny k automatizaci testů ve webovém prohlížeči. Selenium je nabízeno jak pro jazyk Java nebo C#, ale také i pro Python a další.

Selenium webdriver podporuje většinu dnes nejpoužívanějších prohlížečů a to Chrome, Firefox nebo také Opera. Tato knihovna umožňuje přístup k DOMu stránky, díky tomu lze testovat jak jejich funkčnost, tak i správnost dat stránkou zobrazené. Mezi objekty lze vyhledávat jak pomocí id jednotlivých elementů na stránce, ale také jazykem Xpath, popřípadě lze k nalezení elementu použít i javascript.

4.14 JGitFlow

JGitFlow [15] je maven knihovna, která usnadňuje vývojářům práci s gitem. Při manuálním zacházení s větvemi se vývojář vystavuje lidské chybě a také proces práce s větvemi může být leckdy zdlouhavý. JGitFlow tyto problémy řeší automatizací už zavedených postupů a tím eliminuje lidské chyby a díky rychlosti zlepšuje produktivitu.

JGitFlow dokáže vytvářet feature, hotfix a release větve. Samozřejmostí je pak i jejich otagování a spojení do příslušných větví.

5 Popis vlastní práce

V této části bakalářské následuje popis práce, na které jsem se podílel v průběhu mé praxe ve společnosti IDC.

5.1 Projekt Interakce

Interakce je projekt, který má sloužit našim analytikům, kteří komunikují s jejich klienty, aby mohli zaznamenat komunikaci mezi sebou. Tyto interakce mezi analytiky nejsou zdarma a klienti si je musí zaplatit. Je tudíž potřeba mít rychlý, uživatele neobtěžující systém, který bude schopen tyto interakce jednoduše zapsat.

Projekt je napsán v programovacím jazyku Java na serverové části a pomocí javascriptu a rámce AngularJs na klientské části. Klientská část je rozdělena na dvě části, jelikož jeden z požadavků pro vytvoření tohoto projektu bylo, aby také fungoval jako outlook doplněk. Tyto dvě větve klientské strany jsou si z velké části totožné, ale tím že outlook zobrazuje stránky pomocí jádra internetového prohlížeče Internet Explorer, liší se tyto projekty v menších detailech za účelem optimalizace.

Klasická webová aplikace je optimalizována pro webový prohlížeč Chrome.

Součástí projektu interakce je také tvorba reportů dat z již vytvořených interakcí. Tyto reporty čerpají data z PostgreSQL databáze a zobrazují jej uživateli ve formě webové stránky nebo je také uživatel může stáhnout ve formátu Microsoft Office Excel.

5.2 Obrana proti XSS útoku

Úkolem bylo opravit chybu, která se našla v jednom z reportů. Tato chyba umožnila spustit skript, který uživatel zadal jako krátký popis interakce při jejím vytváření.

V reportech tento krátký popis slouží také jako link na vytvořenou interakci. Díky této funkcionalitě v tomto případě nebylo provedeno zakódování znaků. Oprava této chyby byla jednoduchá a spočívala pouze v tom, že před odesláním dat ke zpracování a vytvoření finální html stránky se provedlo zakódování těchto atributů.

Při opětovném testování se ale zjistilo, že tohle není jediný případ provedení XSS útoku na tomto projektu. Hlavním cílem bylo tedy zabezpečit celou aplikaci proti této formě útoku nejlépe tak, aby fungovala nezávisle na jakýchkoliv budoucích změnách v kódu.

5.2.1 Řešení úkolu

Vzhledem k tomu, že je pro nás důležité, aby tato funkcionalita byla nezávislá a při budoucích změnách se nemusela pravidelně upravovat tato část kódu, bylo zapotřebí, aby kód sám mohl reagovat na nové změny bez zásahu programátora.

K řešení tohoto problému jsem využil reflexi. Při jakékoliv změně nebo uložení interakce se na server posílá jeden objekt. Před voláním servisní třídy, která by tyto změny ukládala, volám novou pomocnou třídu, která pomocí reflexe projde všechny atributy jakéhokoliv objektu a pokud je atribut řetězec, tak provede zakódování znaků.

Tento objekt může samozřejmě obsahovat další vnořený objekt. V tomto případě potom metoda volá rekurzivně samu sebe a provede zakódování řetězců i vnořeného objektu. Objekt může taky obsahovat kolekce a i s tímto případem se musí počítat.

Tato pomocná třída se volá jak při ukládání, nebo aktualizaci dané interakce do databáze. V databázi jsou tedy všechny informace zakódovány. Ale také při načítání dat z datábaze. Kdyby byl útočník schopen skripty vložit přímo do databáze, tak jsme tím zajistili, že na straně klienta se data zobrazí správně bez spuštění jakýchkoliv skritpů.

5.2.2 Problémy při řešení úkolu

Při psaní pomocné třídy jsem narazil na několik problémů. Při zanořování do vnořených objektů je potřeba jednoznačně určit zda se jedná o programátorem vytvořený objekt (v podobě nějakého doménového nebo dto objektu) nebo jestli se jedná o primitivní typ, popřípadě jiný objekt. Tento typ objektu jednoznačně určit nelze, alespoň Java k tomuto problému nemá žádnou dokumentaci a neposkytuje žádnou funkcionalitu, která by dokázala jednotlivé objekty takto rozdělit. Všechny objekty lze ale klasifikovat podle jeho balíčku, ve kterém se nachází. Využil jsem tedy reflexe a při vyhodnocování o jaký objekt se jedná, se charakter objektu odvozuje od jména jeho balíčku. Jestli balíček začíná námi definovaným jménem, tak se jedná o programátorem definovaný objekt a může se tedy rekurzivně zanořit a provést kódování znaků.

Tím je vyřešena klasifikace objektů, ale pokud bychom to nechali v tomhle stavu, tak bychom zjistili, že při spuštění nám v aplikaci nastane vyjímka typu "StackOverflowException". Je to způsobeno tím, že jsme nepodchytili všechny stavy. Nemůžeme slepě předpokládat, že se můžeme zanořit do každého objektu, který vývojář vytvořil, a který jsme roztřídili jen na základě jeho balíčku. Existuje totiž speciální třída, do níž bychom se neměli zanořovat a tím je Enum. Enum splňuje podmínku, že se nachází v našem balíčku, ale zanořit se do něj nemůžeme.

Následně nám pomocná třída, která dokáže provést kódování všech řetězců v jakémkoliv objektu, funguje. Další problém nastal při testování na klientovi, kdy jsme předpokládali, že zakódované znaky se v prohlížeči zobrazí normálně. To se tak i běžně děje. Ale jelikož používáme AngularJs a jeho obousměrné vázání dat, které už samo o sobě má podporu proti xss útoku, tak se zakódované znaky nezobrazovaly tak, jak by měly.

Řešení tohoto problému u klasických elementů s párovými značkami bylo jednoduché. Stačilo použít jinou formu vázání dat, která zakódované znaky zobrazí v pořádku. Problém byl při použití u nepárových značek, jako byly v našem případě značky input.

AngularJs podporuje tvorbu tzv. direktiv, jenž umožňují určitou formou měnit nebo přidávat funkcionalitu do určitých částí stránky. Vytvořil jsem tedy direktivu, která pokud se přidá jako atribut do značky input, tak změní způsob vykreslování dat pro tuto komponentu. Změna způsobovala to, že data ukládá do atributu value ve značce input, která implicitně zobrazuje zakódované znaky tak, jak má.

Další problém na který jsem narazil je, že aplikace interakce má také formulář pro popis interakce, ve kterém fungují obecně všechny značky. Důvodem toho bylo, že když uživatel vyrábí interakci z outlooku, může tam upravovat styl psaní (tučné písmo, kurzíva, barva textu atd.). Jenže náš helper provedl zakódování i nad tímto atributem.

Vytvořil jsem tedy vlastní anotaci, kterou jsem následně přidal na atribut popis interakce v našem doménovém objektu. Jakmile na tuhle anotaci narazí naše pomocná třída, tak se neprovede klasický escape. Přidal jsem do projektu Interakce knihovnu OWASP, která umožňuje smazání všech nebezpečných značek (ale i atributů, jako je například onChange), ale dokáže ponechat značky upravující formátování nebo vzhled textu.

5.3 Sekce Klient info

V sekci klient info 3 uživatel zadává jména klientů, se kterými během interakce komunikoval. Sekce klient info musí umět napovídat uživateli podle toho, co už zadal za jméno klienta. Hlavními kritériemi je křestní jméno a příjmení. V případě, že už uživatel zadal nějaké křestní jméno, nápověda v příjmení mu napoví jen takové klienty, kteří mají toto křestní jméno.

Hlavní podmínkou bylo také to, že při výběru určitého klienta z nabídky, se při pohybem myši nebo pomocí šipek budou přepisovat všechna pole v sekci klient info, aby uživatel mohl vidět, co vybírá. V případě, že by ale nic nevybral, musí se sekce klient info vrátit do původního stavu.

Pří ukládání nového klienta, tj. takového kterého uživatel nevybral z nápovědy, se musí počítat s migrací onoho nového klienta do systému Salesforce. To je potřeba, protože následně při celkové migraci všech interakcí do Salesforce, interakce musí mít už v tomto systému vytvořeného toho nového klienta, aby k němu mohla vytvořit vazbu. Salesforce samozřejmě podporuje vytváření nových klientů, ale v případě že by se to dělo až při celkové migraci, tak by toto napojení nešlo udělat. Při vytváření nového klienta databázově, tj. pomocí vkládání záznamu do tabulky, Salesforce nevytvoří jeho unikátní id hned, ale až za nějakou dobu, což je nežádoucí.

Nový klient se bude vytvářet na straně Salesforce pomocí REST volání, kde jako odpověď dostaneme jeho unikátní id.

5.3.1 Řešení úkolu

Jako prvek na stránce jsem využil "ui-select"z rámce AngularJs, který umožňuje podobné zobrazení jako "auto-complete"prvek.

Při posílání požadavku na server se posílají dvě informace, křestní jméno a příjmení. Díky tomu jsme schopni vytvořit propojení mezi těmito informacemi a použít je jako podmínku při přijímání dat z databáze.

Dotaz na databázi může v některých případech, pokud má účet mnoho zaměstnanců, trvat delší dobu. Bylo tedy použito načítací kolečko v každém prvku v sekci klient info, aby uživatel věděl, že se něco děje a dostal zpětnou vazbu.

Přepisování polí, které už uživatel vyplnil při vybírání klienta z nápovědy, se děje pomocí ukládání původního klienta do proměnné při prvotním volání na server. Nemůžeme ukládat do této proměnné informace při každém volání, protože by se nám mohlo stát, že bychom si přepsali už jednou uloženého klienta.

Vrácení do původního stavu, kdy uživatel žádného klienta nevybral, se děje tehdy jakmile prvek, se kterým pracuje, ztratí aktivní zaměření.

Při vytváření nového klienta se volá REST API Salesforce systému. Před tímto voláním je třeba autorizovat novou relaci. Tato autorizace se provádí pomocí OAuth2 autorizačního rámce, která umožňuje získat limitovaný přístup do aplikace přes protokol http. Následně se na server pošle nový kontakt společně s přístupovým žetonem. Odpověď ze serveru obsahuje kód odpovědi. Při úspěšném uložení se jedná o kód 200 nebo 201. V těle zprávy se poté nachází unikatní id nového klienta.

Nově vytvořeného klienta na straně Salesforce zapíšeme i do naší databáze a to i se salesforce id, aby později při migraci bylo možné nového klienta v jejich databázi najít.

Obrázek 3: Výsledná podoba sekce klient info

5.3.2 Problémy při řešení úkolu

Použití prvku "ui-select"pro tuto datavou vazbu mezi křestním jménem a příjmením se ukázalo být dost problematické. Tento prvek jsem musel použít za účelem zachování konzistence v naší aplikaci.

Jako jeden z požadavků bylo, aby při najetí myši na zobrazenou nabídku možných klientů, kteří odpovídají kriteriím zadané uživatelem, se všechny prvky patřící pod jednoho klienta (viz obrázek) mají přepsat na toho, na kterém je nyní ukazatel myši. Tahle funkcionalita má ale také fungovat pomocí šipek. V praxi to znamená, že hned při zobrazení nápovědy je vybrán první klient z nabídky. Při najetí myši na nabídku klientů má kurzor přednost před vybraným klientem a poloha předvybraného klienta zůstane stejná i při vyjetí myši z nabídky. Byl tam také menší problém při vybírání klienta pomocí šipek. Ve výchozím nastavení je předvyplněný klient zbarven jemně šedou barvou, ovšem při použití myši je tato barva jasně modrá. Obejít to šlo zásahem do vnitřních proměnných prvků "ui-select"a při pohybu pomocí šipek měnit index předvybraného klienta.

Výchozí chování na speciální klávesy je vybrání předvybraného klienta, což ale není žádoucí, jelikož při stisknutí klávesy Tab, by se klient vybral. Pomocí atributu jsem vypnul všechny klávesové funkce na tomto prvku a ručně jsem poté naprogramoval vybrání klienta pomocí klávesy Enter a vrácení k původnímu kontaktu pomocí klávesy Tab.

Ui-select prvek jako takový nepodporuje javascript zpětné volání při ztrátě aktivního zaměření. Toto zpětné volání při ztrátě zaměření je potřeba k navrácení původního klienta v případě, že uživatel žádného nevybral.

Ui-select toto nahrazuje svým vlastním zpětným voláním, které se volá, jakmile se prvek uzavře. To znamená jakmile se nabídka nápovědy zavře, zavolá zpětné volání.

Toto zpětné volání funguje správně do té doby, než uživatel klikne z jednoho prvku "ui-select"do druhého prvku "ui-select". V tomto případě se zpětné volání nezavolá. Řešit to lze zavedením značek, které určují zda je určitý prvek otevřený nebo zavřený. Připojením dalšího zpětného volání, které se volá při otevření ui-select prvku, jsme schopni docílit případné korektury mezi těmito značkami. V praxi to znamená, že když je značka křestní jméno otevřená a zavolá se zpětné volání otevření prvku příjmení, nastala právě situace přechodu mezi ui-select prvky a značka křestní jméno se přepíše na zavřená.

Díky tomu jsme schopni vracet zobrazení původního klienta při ztrátě zaměření prvku "ui-select".

Jelikož volání na server kvůli nápovědě a zobrazení klientů je asynchronní, při velice rychlém zadávání je možné, aby uživatel poslal požadavek na server jak z prvku křestní jméno, tak i z prvku příjmení. Jakmile by odpovědi na tyto požadavky přišly zpátky na stranu klienta, tak by se nápověda zobrazila pro každý prvek, i když onen prvek neměl aktivní zaměření. Toto je výchozí chování prvku "ui-select"a nepřišli jsme na to jak tohle chování změnit.

Tento problém jsem vyřešil tak, že jsem při posílání požadavku na server z jednoho prvku,

zakázal poslání požadavku z jiného prvku, dokud nepřišla odpověd ze serveru.

5.4 Posílání emailu při vytvoření interakce

Uživatel může při vytváření interakce přidat delegáta, to je člověk, který zodpovídá za určitý okres. Tito delegáti jsou uloženi v databázi. Uživatel poté může vybrat z přednastavených výběrů delegátů, kteří se poté automaticky přidají do pole pro delegáty.

Pole pro delegáty může uživatel libovolně upravovat. Mazat vybrané delegáty, pokud si vybral z přednastaveného výběru nebo přidávat své vlastní.

Při ukládání interakce se po uložení do databáze pošle email všem delegátům s přehledem o uložené interakci.

Delegátům by neměly chodit neustálé emaily, pokud uživatel stále aktualizuje danou interakci. Email se má poslat jen při uložení interakce a poté jen jednotlivě, pokud by někdo v aktualizaci přidal nového delegáta.

Do interakce lze také přidávat přílohy. Pokud je příloha přidána, v emailu by měla být také přiložena.

5.4.1 Řešení úkolu

Řešení úkolu zahrnovalo vytvořit nové databázové tabulky:

- Seznam přednastavených delegátů.
- Ukládání historie poslaných emailů.

Po uložení nebo aktualizaci interakce do databáze proběhne kontrola, zda byl všem delegátům poslán email. Všem, kterým email poslán nebyl, se vytvoří šablona emailu. Email je vytvořen jako zjednodušená stránka celých interakcí už s předvyplněnými informacemi.

Pro vytváření této šablony byla použita technologie Freemarker. Tato technologie umožňuje vyvořit šablonu s definovanými proměnnými, které posléze při běhu aplikace dosadí.

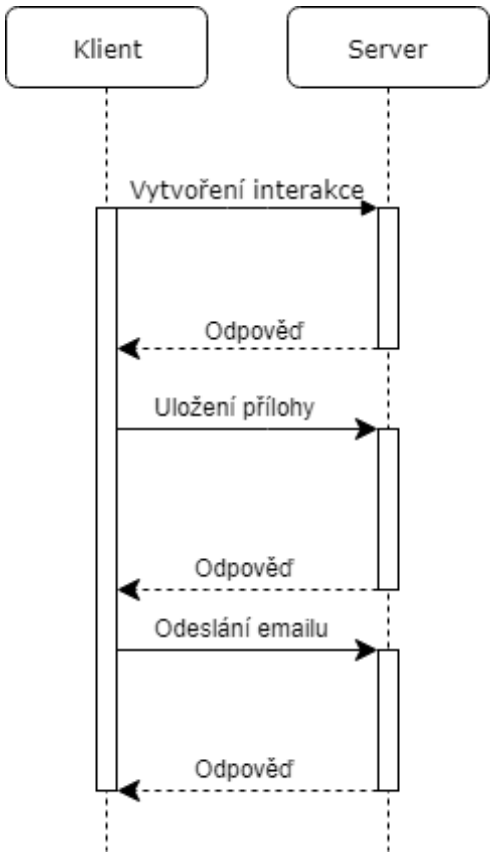
Tato stránka je napsána v html a stylizačně se podobá aplikaci.

Vzhledem k tomu, že interakce může obsahovat také citlivá data, nejsou proto pole s těmito daty vyplněny, ale místo toho odkazují na stránku aplikace. Uživatel, který dostane tento email, nemusí nutně mít práva pro zobrazení dané interakce. Při přesměrování na stránku webové aplikace se už o rozhodnutí, zda uživatel danou interakci může vidět, postará logika aplikace.

5.4.2 Problémy při řešení úkolu

Posílání příloh na server k uložení se provádí až při úspěšném uložení interakce do databáze a příslušné odpovědi ze serveru ke klientovi. Až poté se posílá příloha interakce, aby bylo zaručeno, že daná interakce už v databázi existuje.

Logika posílání emailu se tudíž musela posunout až za uložení přílohy do databáze. Komunikaci mezi klientem poté můžete vidět na obrázku 4.



Obrázek 4: Komunikace klienta se serverem

5.5 Přesun databáze do AWS

Tým (Sage), který má na starost hlavní stránku naší společnosti, přesouval své databáze na servery společnosti Amazon.

Naše aplikace z těchto serverů denně odebírají data. Některé aplikace jsou také zdrojem pro servery Sage týmu. Pro tuto změnu je potřeba upravit všechny transformace pro výměnu dat, které mezi těmito servery fungují.

5.5.1 Řešení úkolu

Tyto transformace obstarává naše aplikace Kettle server. Tato aplikace funguje na bázi kroků, které používají jazyk SQL a je tímto schopna přelévat data z naší PostgreSQL databáze do Sage Oracle databáze a naopak.

Při zásahu do těchto transformací jsme rozhodli, že změníme i způsob synchronizace dat mezi těmito systémy. Předešlé transformace spoléhali na vlastní sloupce v tabulkách, které se měnily podle toho, zda migrace proběhla.

Nový způsob závisí na sloupci poslední změny, který musí být ve zdrojové, ale i v cílové tabulce. Při spuštění transformace se z cílové tabulky vezme maximální hodnota poslední změny. Poté se ze zdrojové tabulky berou všechna data, která mají větší datum poslední změny než maximální datum z cílové tabulky. Tímto je zaručeno, že při každém spuštění transformace se ze zdrojové tabulky nezávisle na cílové tabulce vezmou jen ta data, která v cílové tabulce chybí.

Díky tomuto způsobu migrace dat je docíleno zjednodušení jednotlivých transformací, protože můžeme vypustit aktualizování zdrojové tabulky, zda se určitý řádek promigroval do cílové tabulky.

Díky změnám ve struktuře tabulek musí být také upraveny kontroly dat tabulek. Tyto kontroly se spouští jednou denně a kontrolují počty záznamů v cílové proti zdrojové tabulce. Kontroluje se také individuálně pro každou tabulku jednotlivé sloupce, zda data jsou stejná.

5.5.2 Problémy při řešení úkolu

Protože databáze byla přesunuta ze serverů naší společnosti, dramaticky se zvýšila latence přístupu k těmto datům. Některé transformace přesouvají denně tisíce záznamů a transformace, co předtím zabírala 5 minut, teď zabírá i více než 30 minut.

Jedním z řešení bylo snížit počet přenášených dat. Některé transformace vůbec nezávisely na synchronizaci mezi prostředímy, ale každý den smažou všechna data v tabulce a nahrají je zpátky. Tento způsob nahrávání dat nebyl problém dřív, když byla nízká latence.

Přesun záznamů z tabulky i s více než milionem záznamů netrval déle než minutu. Mezi výhody těchto transformací patří jejich jednoduchost, nepotřebují žádné signalizační vlajky, zda záznam byl zmigrován, ale především kopírují každý den celou tabulku. Když se data ve zdrojové tabulce pravidelně mažou, tyto změny se díky transformaci, která každý den migruje celou tabulku, promítnou i v cílové tabulce.

Kdybychom tuto inicializační transformaci chtěli změnit na inkrementální, která by používala datum poslední změny k synchronizaci dat, museli bychom také vyžadovat změnu aplikační logiky. Data, která se předtím mazala ze zdrojové tabulky, by se teď musela přidávat do nové speciální tabulky, která by potom transformace načítala a mazala z cílové tabulky.

Samozřejmě by to šlo také vyřešit přidáním sloupce, zda záznam byl smazán. Ale to by způsobilo, že záznam, který neměl v tabulce být, tam přetrvává a tabulka si drží svou historii. Problém může nastat, kdyby aplikační logika závisela na tom, že když se záznam smaže, tak už se v dané tabulce nebude dále objevovat.

Ty transformace, které šly změnit na inkrementální, jsme tedy změnili, ale některé na inkrementální změnit nešly.

Tyto transformace byly upraveny tak, aby jeden výběr dat z databáze mohl být spuštěn paraleně několika vlákny. Ke spuštění výběru z databáze v několika vláknech je potřeba, aby zdrojová tabulka měla unikátní identifikátor (id), kterým je číslo. Předpokládám, že i kdyby tento identifikátor nebyl číslo, tak vhodným převodem by šlo docílit stejného výsledku, ale během mé práce jsem se s tímto případem nesetkal.

Rozdělení výběru z databáze na vlákna je provedeno přidáním podmínky do výběru, která říká, že unikátní identifikátor modulo celkovým počtem vláken se rovná číslu právě běžícího vlákna.

- id % počet_vláken = číslo_vlákna

Spuštěním výběru dat s touto podmínkou docílíme jejího rozdělení na jednotlivá vlákna. Při testování jsem zjistil, že zrychlení výběru touto metodou je lineární až do hranice osmi vláken pro jeden výběr. Při přidávání dalších vláken už efektivita postupně klesá. Při 12-ti a více vláknech už k žádnému zrychlování nedochází, ale vyšší režie nových vláken způsobuje postupné zpomalování celé transformace.

Postupným testováním jsem nakonec dospěl k závěru, že nejefektivnější bylo spouštět výběry v osmi vláknech. Přidáváním dalších vláken nezpůsobovalo o tolik rychlejší vykonávání. Zároveň při více vláknech zbytečně vytěžovali procesor serveru, kde databáze běží.

5.6 Migrace ze Salesforce

Aplikace Interakce je zamýšlena jako náhrada za Salesforce systém vytváření interakcí. Interakce, které se ale vytvořily v systému Salesforce, musí být uloženy také v aplikaci Interakce.

Během prvních několika měsíců při oficiálním spuštění aplikace Interakce bude moct vytvářet interakce v obou systémech. Mezi těmito systémy tudíž musí probíhat synchronizace dat.

5.6.1 Řešení úkolu

Salesforce používá vlastní databázi SOQL, která se podobná tradičním SQL databázím. Pomocí nástroje Kettle jsme schopni přelévat data z SOQL databáze do naší PostgreSQL databáze.

V databázi Salesforce jsou data uložena odlišným způsobem a struktura databáze se nijak nepodobá struktuře v naší databázi. Vzhledem k těmto odlišnostem je zapotřebí využít mapovacích

tabulek, které převedou Salesforce data v podobě, v jaké jsou uložena do struktury v databázi Interactions. Lze si to představit jako mapování řetězce ze Salesforce na cizí klíč, který se v Interactions databázi používá místo řetězců.

Naším cílem při vytváření této migrace bylo, aby jedna transformace fungovala zároveň jako inicializace, která nahraje všechna data ze zdroje do cíle. Ale aby sloužila také jako inkrementální transformace, která bude přelévat jen dosud nesynchronizovaná data ze zdroje do cíle.

Vzhledem k našim požadavkům jsem zavedl do našich transformací proměnnou typu boolean, zda se jedná o inicializaci. Díky této proměnné se poté transformace dělí na dvě části. Jedná se o inicializační část a inkrementální část. Inicialiční proměnná na začátku určuje, která část se bude vyhodnocovat.

Tato proměnná se deklaruje před spuštěním všech migračních transformací v aplikaci Kettle server, která obstarává jejich spuštění.

Hlavní výhodou používání jedné transformace pro inicializaci, ale také pro inkrement, je její spolehlivost. Běžně se totiž denně spouští jen inkrementální transformace. Při úpravě těchto transformací se může zapomenout upravit také inicialiační transformace, která ani nemusí existovat. Tím že máme inicializaci i inkrement v jedné transformaci, máme vše na jednom místě a aktualizované na nejnovější podobu struktury databáze.

Jedním z rozdílů SOQL databáze oproti ostatním databázím je, že uživatel, pod kterým se do databáze lze přihlásit, může mít aktivní jen omezený počet připojení. Toto omezení se netýká jen počtu připojení v jeden moment, ale také určitých prvků, jako jsou připojování tabulek do výběru nebo využívání podmínek ve výběru. Vzhledem k těmto omezením a složitosti Salesforce databáze, přeléváme tabulky, které potřebujeme 1:1, do databáze Interakcí. Následně nad těmito připravenými tabulkami provádíme složitější dotazy.

5.7 Automatické testy

Používání aplikace lze shrnout do několika postupů, které uživatel s největší pravděpodobností použije, aby došel k výsledku. Tyto postupy lze zapsat do jednotlivých, po sobě jdoucích, kroků, které se musí vykonat a jsou stále stejné.

Ideálně je poté tento soubor postupů neustále testován člověkem, který se zabývá testováním dané aplikace.

Automatické testy jsou schopné tyto postupy zautomatizovat, pravidelně testovat už zavedené postupy a náhlasit chyby v případě, že se nelze pomocí postupu dostat k výsledku, který byl definován.

Ke tvorbě automatických testů jsme použili programovací jazyk Java a technologii Selenium, která dokáže přistupovat k elementům na webové stránce a pomocí těchto elementů se navigovat na webové stránce.

5.7.1 Porovnání QA a PROD prostředí

Naše aplikace každý den migrují data z PROD prostředí do QA prostředí aplikace. V praxi to poté znamená, že QA prostředí aplikace by mělo mít stejné data oproti PROD prostředí. Lišit by se tedy měly jen kódem, který vývojáři vyvíjejí a je spuštěný na QA prostředí. Kontrola těchto dvou prostředí, kdy předpokládáme, že data by měla být stejná, patří k jedním z jednodušších scénářů, které lze otestovat.
Porovnávací testy by byly časově náročné, kdyby je měl provádět člověk. Proto jsme vyvinuli kontrolu těchto dvou prostředí pomocí automatických testů. Tyto testy dokážou porovnat data za zlomek času, než by zvládl člověk.

5.7.2 Řešení úkolu

K vyhledávání elementů na webové stránce používáme jazyk XPath, který umožňuje vytvořit jedinečnou adresu webového elementu.

Selenium knihovna dokáže pomocí XPath cesty najít jakýkoli element na stránce a použít jej dle potřeby. Například přečtení hodnoty elementu nebo atributu, popřípadě lze využít jednoduchých funkcí jako je například kliknutí.

Důležité je, že stránky jsou více či méně dynamické a málokdy úplně statické. Knihovna Selenium pracuje s web elementy jako s odkazy na element.

V případě, že se jakákoliv část stránky změní, tato cesta nemusí už odpovídat elementu, který Selenium načetlo. Z toho vyplývá, že se k těmto web elementům také musíme chovat jako k adresám. Jestliže chceme data uvnitř adresy, musíme je okamžitě vzít, ale samotnou adresu potom můžeme zahodit, jelikož je pravděpodobné, že na této adrese nemusí být daný element napořád.

Na začátku se test musí přihlásit pod testovacím uživatelem. Uživatel musí mít práva, aby mohl vidět data v aplikaci, ale nesmí to být administrátor, jelikož by to bylo potenciální bezpečnostní riziko.

Poté test jednotlivě prochází všechny stránky a podstránky ve webové aplikaci a ukládá si do paměti všechny informace, které mu webová stránka poskytla.

Jakmile test nashromáždí všechna data, která podle scénáře měl najít, odhlásí se. Po odhlášení se přihlásí do produkčního prostředí a celý proces zopakuje. V případě, že vše proběhlo v pořádku a test je odhlášený z aplikace, provede se porovnání uložených dat mezi sebou. Pokud data stejná nejsou, test si vynutí ukončení. Náš systém Jenkins tyto testy monitoruje a kdyby nastalo vynucené ukoneční testu, pošle automatický email s hlášením o chybě.

6 Zhodnocení

Práce ve společnosti IDC se nedá přirovnat k ničemu, co mi škola jako taková mohla nabídnout. Práce v mezinárodní společnosti, kde každodenní používání anglického jazyka je vyžadováno a možnost vývoje na reálných projektech, které musí projít všemi cykly je něco, co škola neumožňuje. Z tohoto pohledu je vypracování bakalářské práce formou praxe výborná možnost, jak se připravit na kariéru v oboru IT.

Rád bych ale zdůraznil použitelnost některých znalostí, které jsem získal během svého studia. Konkrétně se jedná o předměty Algoritmy 1,2 programovací jazyky 1,2, ale také Úvod do databázových systému a Databázové a informační systémy. Informace, které jsem z těchto předmětů získal, byly nedílnou součástí práce, kterou jsem ve společnost IDC vykonával. Některé znalosti a dovednosti bohužel mé studium neobsahovalo, jako například použití verzovacích systémů GIT a SVN, nebo lepší zastoupení programovacího jazyku Java a její webové nadstavby SPRING. Je škoda, že tyto technologie výuka na škole neobsahuje, jelikož se domnívám, že pro praxi jsou přínosné. Tyto znalosti jsem si musel doplnit samostudiem a dle mého názoru jsou to věci, které se hodí umět na začátku budování kariéry.

7 Závěr

Vypracování bakalářské práce formou praxe ve společnosti IDC byla nenahraditelným milníkem mého začínajícího kariérního života.

Díky praxi, kterou mi umožnila společnost IDC absolvovat, jsem se zdokonalil v mnoha praktických dovednostech. Uvítal jsem možnost samostatné práce pod odborným vedením mnoha programátorů, kteří už mají dlouholeté zkušenosti. Přesto ke mě přistupovali jako k rovnocennému kolegovi. Osobně si cením příležitosti podílet se na vývoji reálného projektu od jeho nápadu, přes návrh, implementaci, testování, až k jeho vydání koncovým uživatelům. Výbornou zkušeností byla také práce v mezinárodním týmu, který dodržuje metodiku vývoje SCRUM. Vyzkoušel jsem si například denní plánování včetně rozboru úkolů, jak budoucích, tak také již vypracovaných. Tento postup je důležitý pro neustálé zlepšování pracovních postupů i efektivity celého týmu. Samostatnou kapitolou byla komunikace se zákazníky, probíhající převážně v angličtině.

Díky oboustranné spokojenosti mi byla nabídnuta další spolupráce se společností IDC, tentokrát již na hlavní pracovní poměr.

Literatura

[1] Idc about. Idc [online]. [cit. 2018-04-14].
Dostupné z: https://www.idc.com/about

[2] Java. Java [online]. [cit. 2018-04-14].
Dostupné z: https://www.java.com/en/download/faq/whatis_java.xml

[3] Java: the complete reference. Ninth edition. New York: McGraw-Hill Education, 2014. ISBN 00-718-0855-8.

[4] About Javascript. MDN Webdocs [online]. [cit. 2018-04-14].
Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

[5] What is maven. Apache maven project [online]. [cit. 2018-04-14].
Dostupné z: https://maven.apache.org/what-is-maven.html

[6] Flyway about. Flywaydb [online]. [cit. 2018-04-14].
Dostupné z: https://flywaydb.org/about/

[7] What Can Querydsl Do for Me. Credera [online]. [cit. 2018-04-14].
Dostupné z: https://www.credera.com/blog/technology-insights/java/can-querydsl-part-1-enhance-simplify-existing-spring-data-jpa-repositories/

[8] Spring. Spring [online]. [cit. 2018-04-14].
Dostupné z: https://spring.io/

[9] Git about. Git scm [online]. [cit. 2018-04-14].
Dostupné z: https://git-scm.com/about

[10] PostgreSQL about. Postgresql [online]. [cit. 2018-04-14].
Dostupné z: https://www.postgresql.org/about/

[11] Sonarqube about. Sonarqube [online]. [cit. 2018-04-14].
Dostupné z: https://www.sonarqube.org/about/

[12] Jira software. Atlassian [online]. [cit. 2018-04-14].
Dostupné z: https://www.atlassian.com/software/jira

[13] Jenkins about. Cloudbees [online]. [cit. 2018-04-14].
Dostupné z: https://www.cloudbees.com/jenkins/about

[14] Selenium introduction. Seleniumhq [online]. [cit. 2018-04-14].
Dostupné z: https://www.seleniumhq.org/docs/01_introducing_selenium.jsp

[15] Jgitflow wiki. Bitbucket [online]. [cit. 2018-04-14].
Dostupné z: https://bitbucket.org/atlassian/jgit-flow/wiki/Home